AD-A057 321 WHARTON SCHOOL PHILADELPHIA PA DEPT OF DECISION SCIENCES F/G 9/2
THE DATA SYSTEM.(U)
1978 K A KIMBALL
UNCLASSIFIED 78-04-03 N00014-75-C-0462
NL

1 OF 1
ADA
057321

END
DATE
FILMED
9-78
DDC

Wharton

Department of Decision Sciences

University of
Pennsylvania
Philadelphia PA 19104

THE DATA SYSTEM

Keith A. Kimball

78-04-03

LEVEL II

Department of Decision Sciences
The Wharton School
University of Pennsylvania
Philadelphia, PA 19104

Research supported in part by the Office of Naval Research under Contract N00014-75-C-0462 and the Burroughs Corporation.

(6) THE DATA SYSTEM.

(10) KEITH ALLEN KIMBALL

(9) Master's thesis,

A THESIS IN

COMPUTER AND INFORMATION SCIENCES

Presented to the Faculty of the Moore School of Computer and Information Science of the University of Pennsylvania in partial fulfillment of the requirements for the degree of Master of Science in Engineering.

(14) 78-04-03 (15) N0014-75-C-0462

(12) 68 p. (11) 1978

Maurice Du Mouchel

Supervisor of Thesis

Graduate Group Chairman

ABSTRACT

THE DATA SYSTEM

KEITH ALLEN KIMBALL

DR. HOWARD LEE MORGAN

This thesis presents the DATA (Dynamic Alerting Transaction Analysis) System as an alternative to a conventional database management system. The DATA System contains no records corresponding to entities but rather is simply a time ordered list of transactions. The advantages of DATA in the areas of security, integrity, and operational ease ~~will be~~ is discussed and the concept of alerting ~~will be~~ is presented. An alerting system provides facilities to monitor changes to the database in order to perform some action whenever certain conditions become true. An alerter can be thought of as a program that continuously monitors the database and takes some specified action when the corresponding condition becomes true. The DATA System is an excellent tool to implement alerting due to its ability to view the database at previous points in time. This work describes the implementation of the DATA System.

ACKNOWLEDGEMENTS

The author wishes to thank his student advisor, Dr. John W. Carr III. Thanks are due to Dr. Rob Gerritsen for suggesting the subject of this work and whose many helpful suggestions are appreciated. Special thanks are due to Dr. Howard Lee Morgan whose encouragement and supervision are greatly appreciated.

Keith Allen Kimball

CHAPTER I

This thesis presents the Dynamic Alerting Transaction Analysis (DATA) System as an alternative to conventional databases. Advantages of this system will be explored and the concept of alerting will be presented. The DATA System will be shown to be an excellent tool to implement alerting. The implementation of this system will be discussed. Applications for the system will be suggested.

Conventional databases are a collection of records used to model certain entities. One physical record corresponds to a given entity and describes the current state of that entity. For example, in a personnel system there could exist a record corresponding to the entity John Smith. This record would contain information such as his name, employee number, current address, and current salary. In the DATA System there exists no physical records corresponding to entities. The database contains no data records as such but rather is simply a time ordered list of transactions. For example, the entity John Smith would be represented by several transactions. The first transaction would correspond to the hiring of John Smith. This transaction would contain his name, employee number, initial salary, and

initial address. As time passed more transactions would affect John Smith. There would be transactions corresponding to him getting a raise and other transactions corresponding to him changing addresses. The current state of the entity can be extracted from this collection of transactions. This current state corresponds to the conventional database record. The DATA database contains more information than the conventional database. Whereas the conventional database provides a static picture of the entity, the DATA database provides a dynamic picture of the entity since it contains all changes the entity has undergone.

DATA databases may be viewed from two levels. On the higher level they appear as ordinary databases. Retrieval, deletion, addition, and modification of records may be performed. An additional feature of a DATA database on the higher level is that it can be viewed as it existed at any previous point in time.

On the lower level, the database may be viewed as a time ordered list of transactions. The three basic types of transactions are deletion, modification, and addition. Once a transaction is entered in the list, it is never modified

or deleted. The format and content of transactions should be well documented so programs can extract from this list and perform some action based on the content of the transaction. The minimal amount of information a transaction should contain is the type of transaction, time of the transaction, a pointer to the previous transaction on the entity, and the new value of any data changed by the transaction. The dynamic database is constantly growing.

An alerting system provides facilities to monitor changes to the database in order to perform some action whenever certain conditions become true. Alerters are the basis of the alerting system. An alerter associates a name (of the alerter), a condition to be evaluated , and an action to be taken when the condition is met. An alerter can be thought of as a program that continuously monitors the database and takes some specified action when the specified condition becomes true.

For example, in a database containing information about airline reservations, an alerter called FLIGHTFULL could be used to monitor the number of available seats on a flight in order to print a message when no seats are available. In this example the alerter name is FLIGHTFULL, the alerting

condition is the number of available seats becoming zero, and the associated action is the message stating the flight number and the fact no seats are available.

The concept of the DATA System has appeared in other systems. Many of the ideas for this thesis came from the similarity found between DATA and differential databases [3]. A differential file is a small data file containing changes to a large readonly master file. One way of looking at a DATA database is as a differential database with the blocks already dumped as the large readonly master files and all blocks written after the dump as the differential files.

A common practice in the maintenance of databases is to keep a log tape. This log tape contains a history of changes to the database. A DATA database can be viewed as this principle taken to the extreme where the log tape becomes the data of the database.

This type of database stores data in a form quite different than the normal user expects to see it. The normal user expects to see tuples while the database stores transactions used to form the tuples. System R's [9] view uses the concept of users' view of the data being quite different from the physical storage of the data.

Alerters change the concept of a database management system from a passive to an active system. Previously the user/database interface consisted of a subroutine call. The user would query the database system and then wait for a response. The user program and database system did not operate in parallel and the database responded only when spoken to. Under an active database system, user programs and database systems operate in parallel and both are capable of action.

Six general advantages can be realized by DATA databases. These advantages are derived from a similarity to differential files [3]. One advantage is related to security and shows that no information is lost due to update. Three are related to database integrity and show that DATA databases can reduce backup cost and provide relatively fast and easy recovery from loses. The final two advantages are operational and show that DATA databases can simplify software development and provide past views of the database.

No information is ever lost due to update when using a DATA database. The physical database is simply a list of transactions. Once a transaction is entered on the list, it

is never modified. Because the database can be viewed as a series of transactions, it is difficult to hide nefarious activity. All changes to the database are localized to a small area. By viewing the database as a series of transactions, this small area can be scanned for improprieties.

DATA databases provide the ability to place consistency contraints on the database that could not be placed on a conventional database. An example of such a constraint is "accept no more than ten changes from a user within one hour". This constraint can be enforced by viewing the database as a series of transactions.

A DATA database can be easily dumped. The list of transactions are simply appended. The only part of the files which need to be dumped are those added since the last dump. This substantially reduces the time required to dump the database. Since transactions are simply appended, there is no danger in dumping the database while it is online.

Fast recovery from hard loses is provided by DATA databases. If a block on the disk is destroyed, it can be simply reloaded from a backup dump and processing can continue. This is possible because transactions are never

modified once they are entered on the list of transactions. The database could be allowed to run while blocks were being copied. If the bad block was referenced either an exception could be returned or the program would wait for the block to be copied.

Recovery from soft loss is also provided. Occasionally programs incorrectly update the database. To recover from this situation the database is backed up to a point in time before the erroneous program ran. The conventional database management system maintains a log tape of "before images" of changes to the database. Backing up the database to a previous point in time is accomplished by reading the log tape backwards and applying the "before images" until the desired point in time is reached. When the database is being updated, both database and log tape records must be written. When the database is being backed up the log tape is read backwards and changes written throughout the database. DATA provides recovery from soft loss without maintaining a log tape. Backing up the database to a previous point in time is accomplished by simply discarding all transactions after the error occurred.

The development of application programs can be

simplified by a DATA database. At some desired point in time the list of transactions could be branched into two separate lists. One list would constitute a production system and the other list would constitute a developmental system running in parallel. Both systems would share the same base list of transactions. Any program being tested would update the database via the developmental list of transactions. These changes will not affect the production database. Thus programs can be tested against the large central database without affecting the production database.

The database may be viewed as it existed at any previous point in time. By ignoring all transactions since a certain time the database is viewed as it existed at that point in time. This property can be used in an efficient implementation of complex forms of alerting. Another use of this feature is to freeze a program's view of an online database at some point in time. The program would view the database as it existed at that point in time while online processing continued.

CHAPTER II

An alerter associates a name (of the alerter), a condition to be evaluated, and an action to be taken when the condition becomes true. The complexity of the condition determines the utility of the alerting. The simplest (and most primitive) way to implement an alerting system would be to evaluate all conditions associated with alerters after every change to the database and to take the specified action if the condition is true. This is very inefficient especially when one realizes that the condition could be any predicate. This predicate could be time consuming to evaluate. Previous authors [1, 2, 5, 6] have distinguished three levels of alerting conditions. These three levels are simple, structural, and complex alerting conditions.

Simple alerting conditions are the easiest to implement. Simple alerting conditions deal only with one relation and the condition can be evaluated by referencing only the old and new copies of the tuple being changed. An example of a simple alerting condition is "tell me when anyone receives a raise over 25%".

Structural alerting conditions deal with changes in the

structure of the database. These require the monitoring of more than one relation. Consider the following database:

```
PEOPLE RELATION (
    NAME KEY ALPHA 15;
    AGE NUMBER;
    CAR ALPHA 10) ;
CARS RELATION (
    CAR KEY ALPHA 10;
    PRICE NUMBER );
```

An example of a structural alerter is "tell me when a person under 30 years of age owns a car priced over $10,000". This requires that both the CARS and PEOPLE relations be monitored.

Complex alerting deals with a more global view of the database. Previous authors [5] have broken these down into two classes. The first class of complex alerting conditions require the use of time. An example of a time spanning alerting condition is "tell me when a car has three price increases in one year". The second class of complex alerting conditions are statistical alerts. An example of a statistical alerter is "tell me when the average price of a car is over $5000".

In a real world situation it seems desirable to limit the scope of the alerting condition. It is felt that the database system should only support simple alerting conditions. This does not preclude the user from writing programs which would emulate the other forms of alerting conditions. A DATA database will contain all the information necessary for implementing the other two classes of alerting conditions via user programs.

The first reason the alerting condition was limited to simple alerting is due to timing considerations. Previous papers have not fully specified when the triggering of an alerter occurs. It has been stated that the alerter is triggered after the update and thus can not interfere with the update [2]. Alerters will now be triggered before the next update. In a conventional database management system this is a necessity because the evaluation of the alerting condition could be changed by the second update. If alerting conditions were time consuming to evaluate then the next update would have to wait for this evaluation and the database would be very slow. Restricting alerting conditions to simple alerting guarantees a relatively cheap evaluation of the condition.

An efficient implementation of the more complex forms of alerting would require auxiliary structures to be created. For example, consider the alerter "tell me when the average salary of department 46500 is over $25,000". An efficient way to implement this is to create a structure containing the number of people in department 46500 and the sum of their salaries. When a change occurs in the salary domain of a tuple whose department number is 46500 then this structure is updated. If the sum of salaries divided by the number of people is over 25000 then the action is taken. The point is that auxiliary structures are created and they may be very large. For example, if instead of department number being 46500 the request was for any department then a auxiliary structure consisting of a tuple per department would be created. The size of this auxiliary structure would be very large. The declarer of the alerter might not be cognizant of this large structure. This is not a desirable feature of an actual database management system.

Just as it was desirable to limit the condition of an alerter, it seems desirable to limit the action an alerter can perform. Previous authors [2, 5, 6] have stated that the triggering of an alerter should generate a message. In a conventional database management system it is necessary

for the message to contain the value of any domains that the declarer of the alerter might be interested in. This is because once the alerter is triggered the database is being updated and these values might change. In the DATA System the message need only to consist of the alerter name and a pointer to the transaction triggering the alerter. The value of any domains of interest can be extracted from the transaction.

Structural and complex alerting conditions could be implemented by requiring users to write programs that use simple alerters to note changes in the database. The programs would then evaluate the more complicated condition to decide if the action should be taken. These appear as ordinary alerters except they may not occur before the next update. This is because after the simple alerter is triggered, the updating of the database continues. The burden of the decision of how to optimize complicated alerting is shifted from the system to the user. The user makes the storage/processor time tradeoffs, and the user is aware of the auxiliary structures his programs use.

This approach to structural and complex alerting conditions depends upon a list of transactions. After the

simple alerter is triggered, processing continues and the database is undergoing change. While the structural alerting condition is being evaluated, the database needs to be viewed at the time the simple alerter was triggered. DATA provides the ability to view the database at that point in time.

Complex alerting conditions require the ability to view the database at previous points in time. This is provided by the DATA System. Complex alerters are also interested in the changes the entities have undergone. This feature is provided by viewing the database as a time stamped list of transactions.

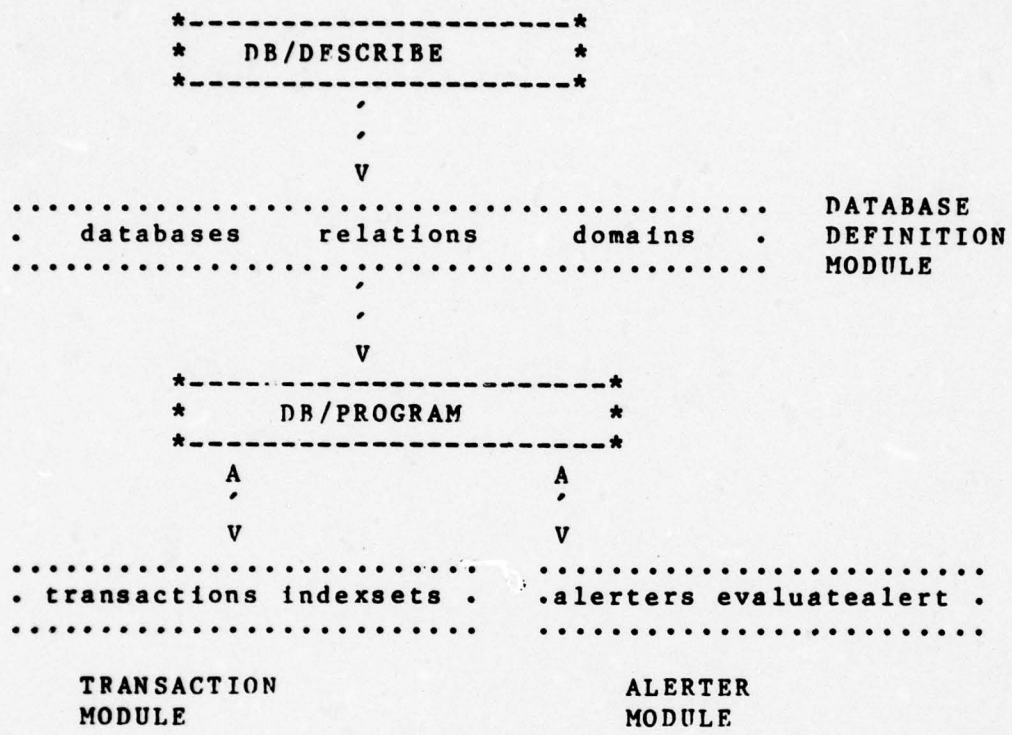
The alerting system proposed restricts alerting conditions to simple alerting conditions and restricts alerting actions to sending a message consisting of the alerter name and a transaction number. The alerting action will occur after the update which triggered it and before any other updates. Users are able to implement structural and complex alerting conditions by writing programs that use simple alerting to note changes in the database and then evaluate the more complicated conditions. The programs require the ability to view the database at previous points

in time and to view the database as a time stamped list of transactions. The DATA System is an excellent tool to implement alerting.

CHAPTER III

This chapter describes the implementation of the DATA System. The database system implemented is a simple relational database system. A database definition consists of a database name, relation definitions, and domain definitions. Access to the database is via a program which provides the facilities of a typical database manipulation language. The commands to access the database are not embedded within a programming language. The commands provided are add, delete, modify, find(at key condition only), show, display, alert, settime, and rollback. The three unusual commands are alert, settime, and rollback. The alert command creates alerters; the rollback command backs the database up in time; and, the settime command allows the database to be viewed at a previous point in time. The system is implemented using a conventional database management system.

An overall picture of the DATA System is:



THE DATA SYSTEM

The database is described by running a program called DB/DESCRIBE. This program prompts the user to describe the database. The first input is the database name. The database name is stored in a relation as follows:

```
DATABASES RELATION (
    DBNAME KEY ALPHA 15;
    DBNUMBER NUMBER);
```

The DBNAME is the database name. DBNUMBER is a number

associated with the database name and is used to identify the database.

The next inputs are the relation names. Relation names are stored in a relation as follows:

```
RELATIONS RELATION (
        RELDBNUMBER KEY NUMBER;
        RELNAME KEY ALPHA 15;
        RELNUMBER NUMBER;
        RELKEYSIZE NUMBER;
        RELSIZE NUMBER);
```

The RELDBNUMBER is the number of the database which owns the relation. The RELNAME is the name of the relation. Relations within a database must have unique names. RELNUMBER is a number assigned to distinguish between relations of the same database. RELDBNUMBER and RELNUMBER uniquely identify the relation. RELKEYSIZE and RELSIZE are the number of characters in a key and in a tuple of the relation.

The final inputs are the domain definitions. Domains are described in a relation as follows:

```
DOMAINS RELATION (
        DOMDBNUMBER KEY NUMBER;
```

```
        DOMRELNUMBER KEY NUMBER;
        DOMNAME KEY ALPHA 15;
        DOMAALPHA NUMBER;
        DOMAKEY NUMBER;
        DOMLOC NUMBER;
        DOMKEYLOC NUMBER;
        DOMSIZE NUMBER);
```

DOMDBNUMBER and DOMRELNUMBER identify the database and relation which own the domain. The DOMNAME is the name of the domain. Domains must have unique names within a relation. DOMAALPHA and DOMAKEY are booleans indicating if the domain is an alpha or if the domain is a key. The DOMLOC and DOMKEYLOC contain the offset of the domain within a tuple and within a key. If the domain is not a key its DOMKEYLOC is zero. DOMSIZE is the size of the domain in terms of characters.

Consider the following database definition:

```
PERSONNEL DATABASE (
    PEOPLE RELATION (
        FIRSTNAME KEY ALPHA 5;
        LASTNAME KEY ALPHA 10;
        ROOM ALPHA 4; ) ;
```

```
ROOMS RELATION (
     ROOM KEY ALPHA 4 ;
     EXTENSION NUMBER ; ); );
```

The database definition would be stored as follows:

DATABASES (prefix column names by DB)

NAME	NUMBER
PERSONNEL	1

RELATIONS (prefix column names by REL)

DB#	NAME	NUMBER	KEYSIZE	SIZE
1	PEOPLE	1	15	19
1	ROOMS	2	4	16

DOMAINS (prefix column names by DOM)

DB#	REL#	NAME	ALPHA	KEY	LOC	KEYLOC	SIZE
1	1	FIRSTNAME	YES	YES	0	0	5
1	1	LASTNAME	YES	YES	5	5	10
1	1	ROOM	YES	NO	15	0	4
1	2	ROOM	YES	YES	0	0	4
1	2	EXTENSION	NO	NO	4	0	12

Appendix B contains syntax diagrams and semantic definitions for a database definition language. This is intended as a logical equivalent to what the user is

prompted to input.

The database is accessed by running a program called DB/PROGRAM. This program manipulates two basic structures. The first structure is called TRANSACTIONS and consists of a time ordered list of transactions. The format of this structure is:

```
TRANSACTIONS RELATION (
        TRANSDBNUMBER KEY NUMBER;
        TRANSNUMBER KEY NUMBER;
        TRANSTYPE NUMBER;
        TRANSRELNUMBER NUMBER;
        TRANSPREVNO NUMBER;
        TRANSYEAR NUMBER;
        TRANSDAY NUMBER;
        TRANSHOUR NUMBER;
        TRANSMINURE NUMBER;
        TRANSSECOND NUMBER;
        TRANSDATA ALPHA 512);
```

The TRANSDBNUMBER identifies the database affected by the transaction. The TRANSNUMBER is a number assigned to distinquish between transactions on a given database. The TRANSDBNUMBER and the TRANSNUMBER identify a transaction.

The TRANSTYPE specifies the type of transaction. The types of transactions are addition, deletion, and modification. The TRANSRELNUMBER is the relation number of the tuple being affected by this transaction. TRANSPREVNO is the TRANSNUMBER of the previous transaction on the entity. All transactions upon an entity are linked via this domain. TRANSYEAR, TRANSDAY, TRANSHOUR, TRANSMINUTE and TRANSSECOND datetime stamp the transaction. TRANSDATA is the new tuple. In practice only the number of characters necessary to contain the new tuple are stored in the database.

The other basic structure manipulated by DB/PROGRAM is called INDEXSETS. The tuples of INDEXSETS are used to locate the most recent transactions upon the entities. This structure provides the head of the linked list of transactions upon an entity. The structure is as follows:

```
INDEXSETS RELATION (
        ISDBNUMBER KEY NUMBER;
        ISRELNUMBER KEY NUMBER;
        ISKEY KEY ALPHA 256;
        ISDELETE NUMBER;
        ISTRANSNUMBER NUMBER);
```

The ISDBNUMBER and ISRELNUMBER identify the database and

relation to which the key belongs. ISKEY is the key uniquely identifing a tuple. ISDELETE is a boolean indicating whether or not there is a current tuple in the database with that key value. ISDELETE is true if a tuple existed with the key value and then was deleted from the database. ISTRANSNUMBER is the TRANSNUMBER of the last transaction upon the entity. This provides the head of the linked list of transactions upon the entity.

Consider the following series of commands:

```
ADD PEOPLE (FIRSTNAME = "JAMES", LASTNAME="CARTER")
MODIFY PEOPLE ( ROOM = "BLUE" )
ADD ROOMS ( ROOM = "BLUE", EXTENSION = 1)
ADD PEOPLE (FIRSTNAME = "BERT", LASTNAME="LANCE")
MODIFY PEOPLE ( ROOM = "RED" )
ADD ROOMS ( ROOM = "RED", EXTENSION = 2)
ADD PEOPLE (FIRSTNAME = "CYRUS", LASTNAME="VANCE")
MODIFY PEOPLE ( ROOM = "PINK" )
ADD ROOMS ( ROOM = "PINK", EXTENSION = 3)
DELETE PEOPLE (FIRSTNAME = "BERT", LASTNAME="LANCE")
```

The database would be stored as follows:

TRANSACTIONS (prefix column names by TRANS)

DB#	#	TYPE	REL#	PREV#	DATA	
1	10	D	1	5		
1	9	A	2	0	PINK3	
1	8	M	1	7	CYRUSVANCE	PINK
1	7	A	1	0	CYRUSVANCE	
1	6	A	2	0	RED 2	
1	5	M	1	4	BERT LANCE	RED
1	4	A	1	0	BERT LANCE	
1	3	A	2	0	BLUE1	
1	2	M	1	1	JAMESCARTER	BLUE
1	1	A	1	0	JAMESCARTER	

INDEXSETS (prefix column names by IS)

DB#	REL#	DELETE	TRANS	KEY
1	1	NO	2	JAMESCARTER
1	1	YES	10	BERT LANCE
1	1	NO	8	CYRUSVANCE
1	2	NO	3	BLUE
1	2	NO	6	RED
1	2	NO	9	PINK

The database system implemented contains ten commands. A simplified description of the implementation follows.

The ADD command creates a new tuple in the database.

This command contains a relation name and initial values for various domains. The first step is to verify that no tuple exists with the same key value as the tuple being created. After this is done a transaction is entered on the list of transactions. This transaction will be of type addition. The datetime stamp is placed in the appropriate domains. The TRANSDATA domain contains the new tuple of the specified relation. The TRANSRELNUMBER is the number associated with the specified relation. The TRANSPREVNO is zero if there was no previous transaction on the entity. If there was a previous transaction on the entity, the TRANSNUMBER of the previous transaction (which must have been a deletion) will be stored in the TRANSPREVNO domain. Finally the INDEXSETS relation is updated. If there was no previous transaction on the entity, a new tuple of INDEXSETS is created. This tuple points to the new tuple of TRANSACTIONS. If there was a previous transaction on the entity, the corresponding tuple of INDEXSETS has its ISDELETE domain reset. This tuple also points to the new tuple of TRANSACTIONS.

The DELETE command deletes a tuple from the database. This command contains a relation name and values to which the key must match. First a tuple is located matching the specified key values. Next a transaction is entered on the

list of transactions with a type of deletion. Finally the tuple in INDEXSETS is marked with ISDELETE as true and pointed to the recently created transaction.

The MODIFY command requires a tuple to be currently located (via a previous FIND or ADD command). This command contains a relation name and new values for nonkey items. The tuple is entered on the list of transactions with the new values of the nonkey items. The corresponding INDEXSETS' tuple is pointed to the new transaction.

The FIND command requires a relation name and values to which keys must match. The program searches INDEXSETS for a tuple matching the specified key values. If no such tuple exists, an exception is returned. If a tuple is found with its ISDELETE domain true and if the database is being viewed from the current time, an exception is returned. Otherwise the corresponding tuple of TRANSACTIONS is read. If the database is being viewed from the current time, the command is complete. If the database is being viewed from a past time, transactions upon this entity are read until one is found with a datetime stamp less than the time that the database is being viewed from. If such a transaction exists and its type is not deleted, the command is sucessful else

an exception is returned stating that there was no such tuple at the specified time.

The SETTIME command allows the database to be viewed from previous points in time. SETTIME=CURRENT states that the database should be viewed from the current time. SETTIME by itself asks the time that the database is being viewed from. SETTIME= 77/155 @ 23:12:11 implies that the database should be viewed as it existed at 11:12:11 p.m. on the 155th day of 1977.

The SHOW command is used to display the relation and domain names. SHOW by itself displays all relation names of the database. SHOW <relation name> displays the domain names of that relation. This is accomplished by reading the RELATIONS and DOMAINS relations.

The DISPLAY command requires a relation name. It causes values in the current tuple of relation name to be displayed at the terminal. For each domain its name and its value within the current tuple is displayed.

The ROLLBACK command backs up the database to a specified time. This is accomplished by simply updating pointers in INDEXSETS. No tuples of TRANSACTIONS are

deleted. The rollback is accomplished by locating each tuple in INDEXSETS and then reading the corresponding TRANSACTIONS tuple. If the TRANSACTIONS' tuple is beyond the rollback point, the linked list of transactions on this entity are read until one is found before the rollback point. If a transaction is found before the rollback point, the INDEXSETS tuple is updated to point at this transaction and the ISDELETE domain is updated based on the type of transaction. If no transaction is found before the rollback point, the INDEXSETS tuple is deleted.

The final command implemented is the ALERT command. Alerters in this system are restricted to simple alerting conditions and the associated action is merely a display at the terminal. This command is used to create, display, enable, disable, and cancel alerters. ALERT by itself causes the names of all alerters to be displayed. ALERT <alert name> causes the text and state of <alert name> to be displayed. ALERT <alert name> = CANCEL causes <alert name> to be cancelled. An alerter can be in one of two states. If an alerter is enabled, it is a candidate to be triggered. If an alerter is in a disabled state, the alerter may not be triggered until it is enabled. The alerter may be placed in these states by the commands ALERT <alert name> = ENABLE or

by ALERT <alert name> = DISABLE . An example of the creation of an alerter is:

```
ALERT BIGRAISE (MODIFY OF PEOPLE) =
    SALARY.NEW > SALARY.OLD * 1.1
```

BIGRAISE is the name of the alerter. The MODIFY OF PEOPLE states that the alerter should be evaluated after the modification of the relation called PEOPLE. The condition is SALARY.NEW > SALARY.OLD*1.1 (i.e. a 10% raise). SALARY is assumed to be a numeric domain of PEOPLE. When an alerter is evaluated two tuples can be seen. If the type of transacton is modification, they are the tuple before and after the modification. For transactions of type addition or deletion, the two tuples are identical. In the case of a transaction of type addition, both tuples are the result of the addition. In the case of a transaction of type deletion, both tuples are the tuple before the deletion. Alerters are stored in two structures. One structure contains the alerter definition and the other contains information on when to evaluate the alerter. The alerter definition is stored in a structure as follows:

```
ALERTERS RELATION (
    ALDRNUMBER KEY NUMBER;
```

```
        ALNAME KEY ALPHA 15;
        ALNUMBER NUMBER;
        ALENABLE NUMBER;
        ALTEXT ALPHA 256);
```

ALDBNUMBER is the number of the database to which the alerter belongs. ALNUMBER is a number used to distinquish the alerter from other alerters in a given database. ALNAME is the name of the alerter. ALNAME must be unique within a given database. ALENABLE is a boolean indicating if the alerter is in an enabled or disabled state. ALTEXT is the text of the alerter.

The structure used to determine when to evaluate the alerting condition is as follows:

```
EVALUATEALERT RELATION (
        EVDBNUMBER KEY NUMBER;
        EVRELNUMBER KEY NUMBER;
        EVTYPE KEY NUMBER;
        EVALNUMBER KEY NUMBER)
```

EVDBNUMBER and EVRELNUMBER are database and relation numbers. EVTYPE is a type of transaction (i.e. addition, deletion, or modification). EVALNUMBER is an alerter number.

A tuple is interpreted to say that upon a transaction of the specified type on the specified relation, the associated alerting condition should be evaluated. The routine that creates tuples in the TRANSACTIONS relation checks this structure after creating a transaction. It evaluates the appropriate alerting conditions and triggers the alerter if the condition is true.

Consider the following ALERT commands:

```
ALERT NEWPRES (MODIFY,ADD OF PEOPLE) =
        ROOM.NEW="BLUE"
ALERT NEWPHONE (MODIFY,ADD OF ROOMS) =
        MODIFY AND
        EXTENSION.OLD NEQ EXTENSION.NEW OR
        ADD
ALERT NEWPRES = DISABLE
```

These alerter definitions would be stored as follows:

ALERTS(prefix column names by AL)

DB#	NAME	#	ENABLE	TEXT(same as definition)
1	NEWPRES	1	NO	--
1	NEWPHONE	2	YES	--

EVALUATEALERT (prefix column names by EV)

DB#	REL#	TYPE	AL#
1	1	M	1
1	1	A	1
1	2	M	2
1	2	A	2

DB/PROGRAM is a program written using the techniques of step-wise program composition[7]. The program can be divided into eight modules. Each module has knowledge of only the data structures and procedures declared in a lower level module. The hierarchial ordering of the eight modules are:

OUTPUT

INPUT

SCAN

DATABASE

RECORDS

- locis
- evaluatealert
- storetrans

TIME

UTILITY

- map

SYNTAX

The function of the first three modules (OUTPUT, INPUT, and SKAN) are standard to most interpretors. The DATABASE module retrieves data from the database. The RECORDS module contains three major routines. These routines are LOCIS, EVALUATEALERT, and STORETRANS. LOCIS is passed a relation number. This routine locates a tuple of INDEXSETS containing the same key value as KEYS[relation number]. EVALUATEALERT is a routine passed an alert number and evaluates that alerter based upon the old and new values of the tuple being changed. STORETRANS is the routine that updates the database. This routine maintains the TRANSACTIONS and INDEXSETS relations and decides which alerters are to be evaluated based on the change to the database. The TIME module contains routines to allow the database to be viewed from previous points in time. The UTILITY module contains the MAP routine. This routine is passed a parameter called OP. OP specifies whether keys are allowed, non keys are allowed, the records array should be cleared, the keys array should be cleared, or if the old tuple should be saved. Its basic function is to process the mapping (i.e. RELATION (domain=value,domain=value)) and fill the KEYS, RECORDS, and OLDRECORD arrays based upon the

mapping and value of OP. The SYNTAX module contains a procedure per command. Each procedure calls routines on the lower level to perform that command. An ADD command would proceed as follows.

1. MAP is called to syntax the command and fill the arrays. The value of OP would be clear keys, clear record, keys allowed, and non keys allowed.
2. LOCIS is called to verify that no tuple exists with the same key value as is being added.
3. STORETRANS is called to store the transaction in the database
4. EVALUATEALERT is called for each alerter that is to be evaluated upon the addition to the specified relation. EVALUATEALERT determines if the alerter should be triggered.

Appendix C contains syntax diagrams and semantic definitions for the database manipulation language.

CHAPTER IV

A likely criticism of the DATA System is the amount of storage used. Transactions are never deleted from the database and consequently it is constantly growing. One solution to this problem is to archive transactions before a given time onto a less expensive mode of storage. Eventually these transactions could be discarded and questions about them could not be answered. This is analogous to a typical business procedure. When a memo is received it is usually placed on a desk (fast retrieval device). After the memo is read it is filed away (archived to a less expensive device). Eventually memos from that year are unimportant and discarded. Another solution to the storage problem is to provide a garbage collection algorithm to discard unwanted transactions. This is also analogous to a typical business procedure. When a filing cabinet becomes full, somebody may go through it to consolidate and discard memos. The algorithm used in determining what to consolidate and what to discard is the garbage collection algorithm. Another solution to the storage problem is a possible hardware breakthrough. If mass storage were to become extremely inexpensive, there would be very little concern about the

amount of storage used.

It has been proposed that a tuple consist of two distinct portions. One portion would contain the relatively constant domains and the other portion would contain the dynamic domains. The relatively constant domains and the dynamic domains would be stored separately. The advantages of this approach are derived from the fact a smaller portion of the tuple needs to be copied when a modification occurs. This leads to less storage used and quicker updates. The disadvantage is that assertions about the frequency of modification of the domains are made at database definition time.

The DATA System has properties that make it ideal for many applications. The applications that will be discussed deal with security, history, monitoring changes, online systems, and adhoc inquiries.

The fact there is a complete history of all changes to the database is ideal for security. Viewing the database as a series of transactions would make it easier to detect improprieties. Instead of having a very large collection of records and trying to detect improprieties in them, dynamic databases isolate all changes into a small series of

transactions which can be scanned for improprieties. The impropriety can be viewed from the point in time at which it occurred. Alerters could be used to detect abnormal changes in the database. Once an impropriety has been identified it can be traced to a series of transactions which may help to identify the culprit.

Applications interested in past states of the entities are suited for the DATA System. Many applications fall into this category. Using conventional database techniques the user is forced to simulate time stamped transaction databases. For example, consider a personnel system keeping a history of employee salaries. This information could be stored in a structure as follows:

```
PEOPLE RELATION (
        EMPLOYEENUM KEY NUMBER;
        ADDRESS ALPHA 20;
        NAME ALPHA 20 );

SALARIES RELATION (
        EMPLOYEENUM KEY NUMBER;
        DATE KEY ALPHA 6;
        SALARY NUMBER);
```

The SALARIES relation amounts to a time stamped list of transactions of salary changes. Using the DATA System a natural representation of this model would be:

```
PEOPLE RELATION (
        EMPLOYEENUM KEY NUMBER;
        ADDRESS ALPHA 20;
        NAME ALPHA 20;
        SALARY NUMBER )
```

The DATA System does the work for the user and provides a more natural representation for the data. Any questions that the conventional database can answer can be answered by the DATA System.

The DATA System has been shown to be a practical way to implement alerting. The database can be viewed at past times and thus the database may be updated while evaluation of a complex alerting condition occurs. Two applications that are ideal for monitoring changes via alerting are economic models and inventory control.

Some applications are required to be online for long periods of time and still require reports to be generated. This has lead to awkward models in order that the reports

may be generated with consistent data. The DATA System could discard these awkward models because they provide the ability to freeze a program's view of the database. Thus online updating of the database could continue while the report was being generated.

Some applications deal with adhoc inquiries. These systems are asked to respond to questions they have no apriori knowledge of. A DATA database never discards any information and thus would be best able to answer the inquiry. The information is well stuctured, and this structure is known to the system. A management information system is an example of such a system.

DATA has provided ideas for future research. One project would be to embed the facilities of DATA into a programming language. For DATA to reach its full potential the database system/user program interface should be a message passing system. This means that both the user program and database system would send and receive messages. After a message is sent, the user program (or database system) continues on other work and later handles the response (and any other requests) to the message. Another project could be to investigate techniques to decrease the

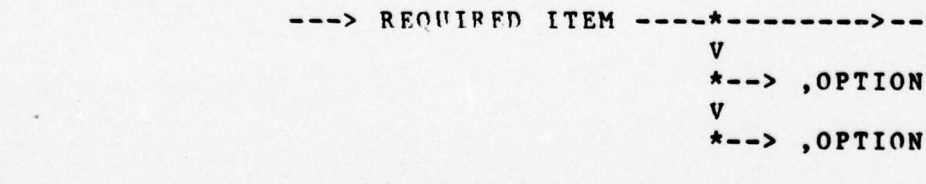
amount of storage used by DATA. Currently DATA supports only three types of transactions. These are ADD, DELETE, and MODIFY. It might be possible for the user to define higher level transactions such as "salary increase". The user would also provide the routines to apply the semantics of such a transaction. Another area of research would be to use DATA to implement more complex forms of alerting. This system would use DATA's simple alerters, ability to view the database as a series of transactions, and ability to view the database at previous points in time to implement the complex alerting.

The DATA System provides an alternative to the conventional database management system. It has advantages in the areas of security, integrity, and operational ease. The DATA System is an excellent framework on which to build an alerting system. An alerting system provides the ability to have the system monitor changes in the database and notify the user when conditions of interest become true. A conclusion of this thesis is that the only type of alerting the system should provide is simple alerting. Users can supply programs that use simple alerting to note changes in the database. These programs evaluate the more complicated alerting conditions. This concept depends upon the feature

of the DATA System which allows the database to be viewed as it existed at previous points in time.

APPENDIX A: RAILROAD DIAGRAMS[8]

Railroad diagrams show how syntatically valid statements can be constructed. Traversing a railroad diagram from left to right, or in the direction of arrow heads will produce a syntatically valid statement. Continuation from one line of a diagram to another is represented by a capital O "O" at then end of the current and the beginning of the next line. The complete syntax diagram is terminated by double slashes "//". A star "*" is used to highlite intersection points. The various arrow heads are left arrow head "<", right arrow head ">", upward arrow head "A" and downward arrow head "V". Items contained in broken brakets "< >" are syntatical variables which are further defined. For example:



produces the following syntatical valid statements:

REQUIRED ITEM

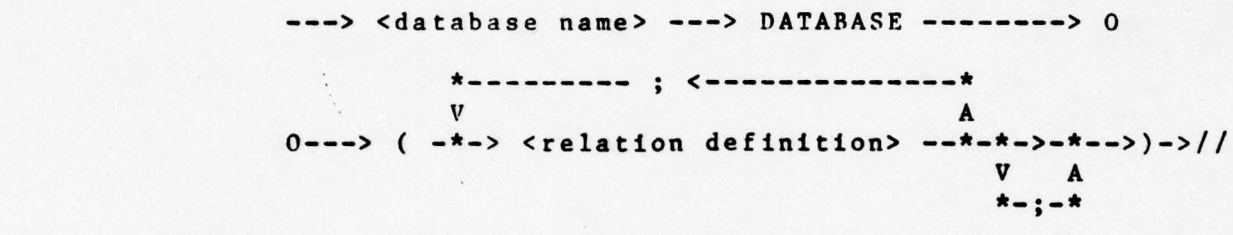
REQUIRED ITEM,OPTION 2

REQUIRED ITEM,OPTION 3

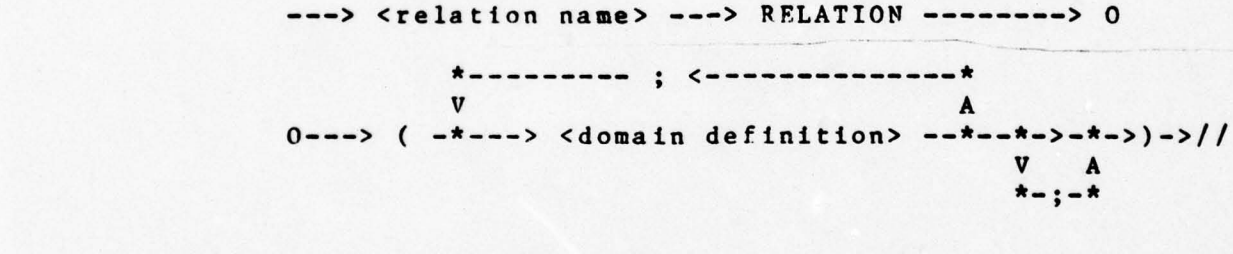
APPENDIX B: DATABASE DEFINITION

SYNTAX:

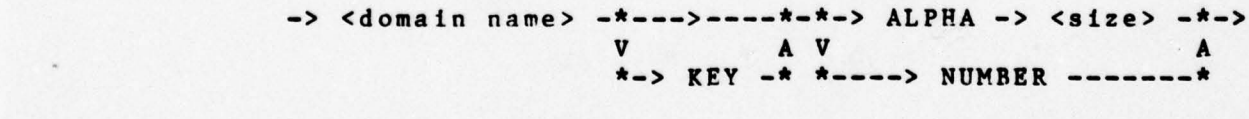
<database definition>



<relation definition>

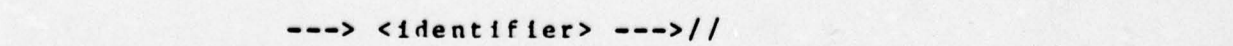


<domain definition>



<database name>

<relation name>



<domain name>

---> <identifier> --->//

<size>

---> <integer> --->//

SEMANTICS:

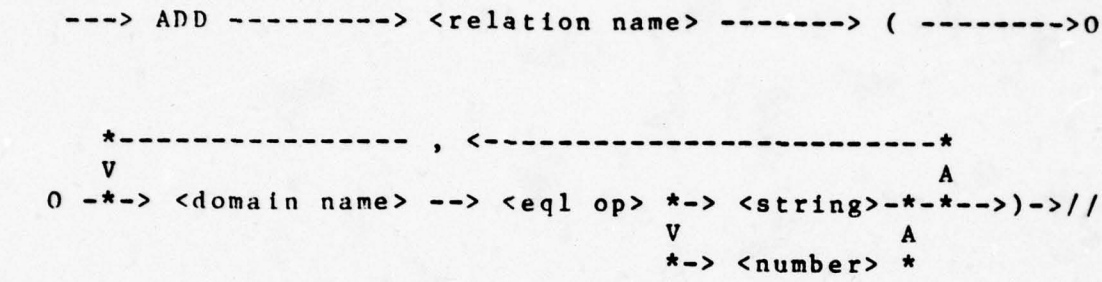
1. A database consists of an identifier which identifies the database and a series of relation definitions. Each relation must have a unique name from other relations.
2. A relation definition consists of a series of domain definitions. Each domain must have a unique name from the other domains within a given relation. At least one domain in every relation must be specified as a key. Keys of a relation uniquely identify a tuple. In other words, no two tuples can have the same key values.
3. Two types of domains are allowed. Domains of type NUMBER are used to store numbers and domains of type ALPHA store strings of length <size> characters.

APPENDIX C: DATABASE MANIPULATION

The DATABASE MANIPULATION LANGUAGE consists of ten commands used to access the database. These commands are the add, alert, bye, delete, display, find, modify, rollback, settime, and show commands.

<add command>

SYNTAX:



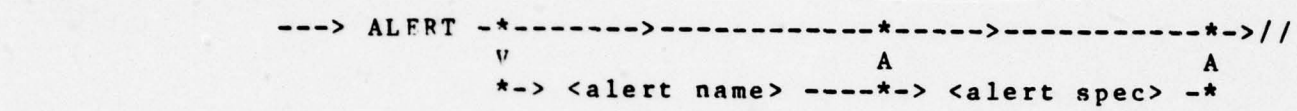
SEMANTICS:

1. The <add command> causes a tuple of <relation name> to be created and stored into the database. An exception is returned if a tuple already exists with a key value identical to the specified values.
2. The value for any alpha domain not mentioned is blank. The value for any number not mentioned is zero. Both keys and nonkeys can be used for <domain name> in a <add command>.
3. Domains of type ALPHA must have <string>s assigned to them and domains of type NUMBER must have <number>s assigned to them.
4. The tuple stored becomes the current tuple of <relation name>.

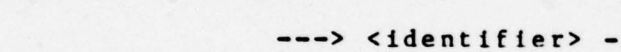
<alert command>

SYNTAX:

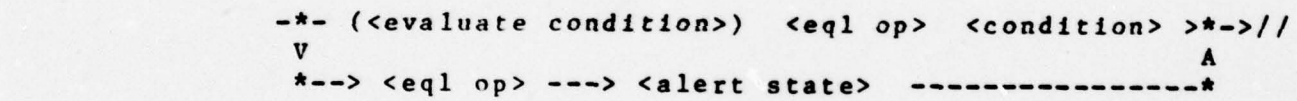
<alert command>



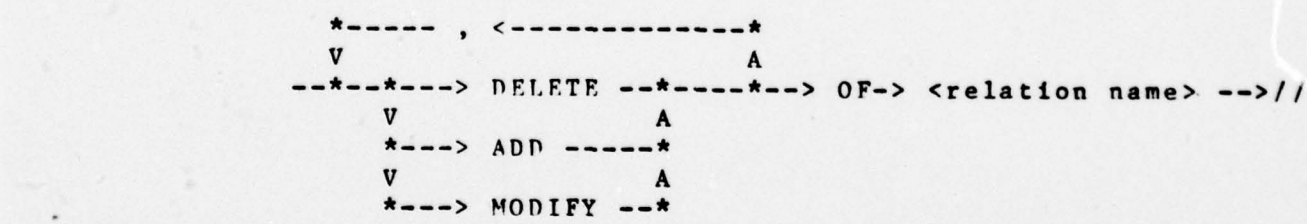
<alert name>



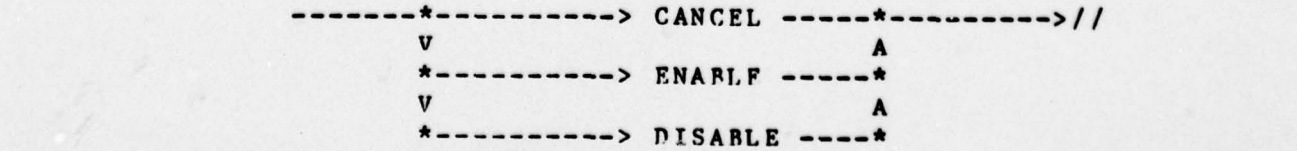
<alert spec>



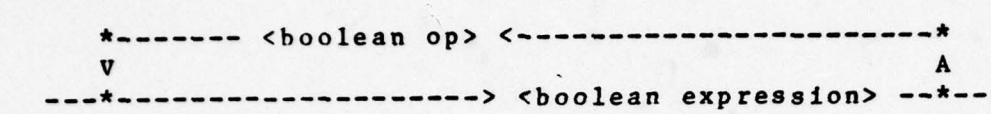
<evaluate condition>



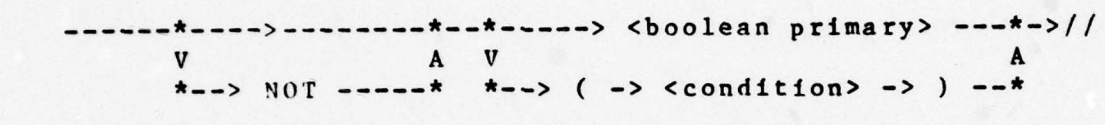
<alert state>



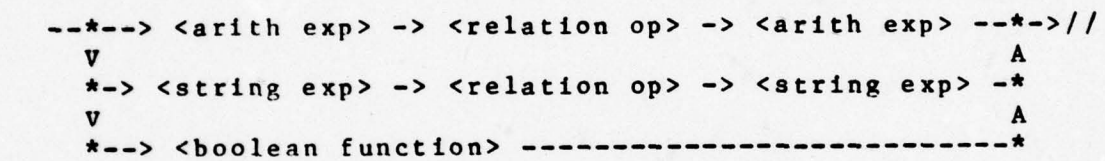
<condition>



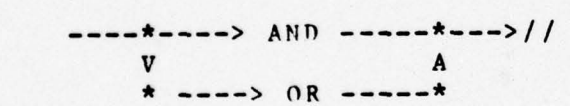
<boolean expression>



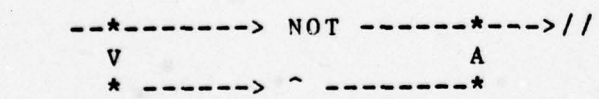
<boolean primary>



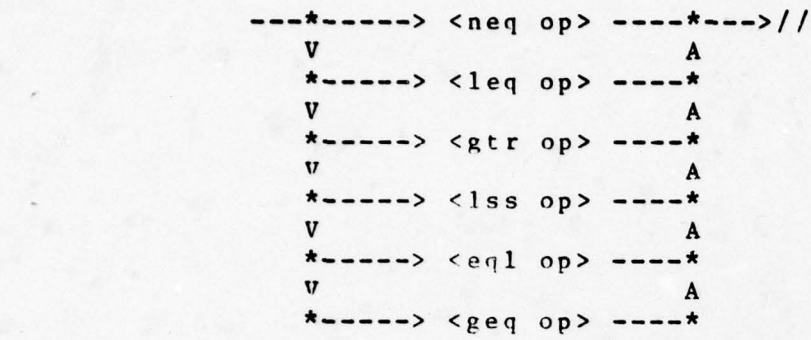
<boolean op>



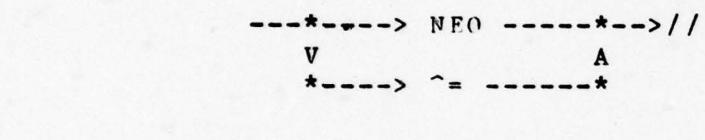
<not op>



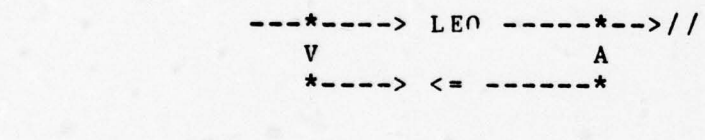
<relation op>



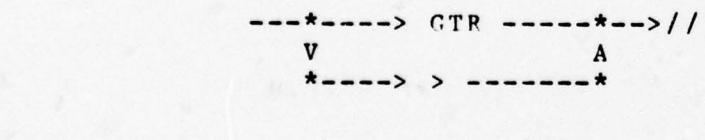
<neq op>



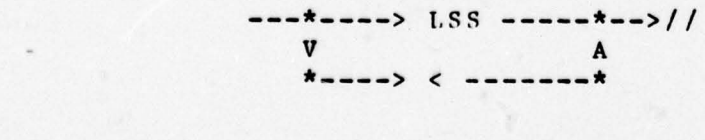
<leq op>



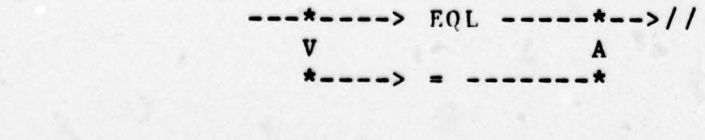
<gtr op>



<lss op>

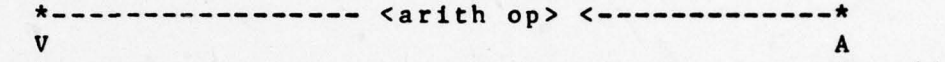


<eql op>

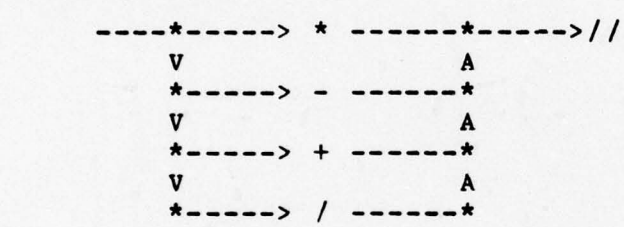


<geq op>

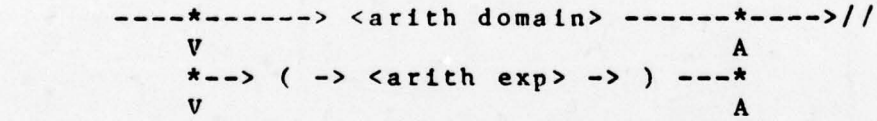
<arith exp>



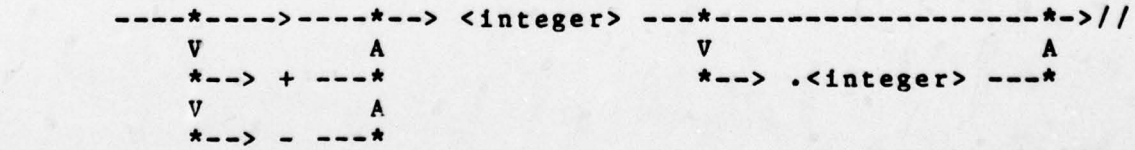
<arith op>



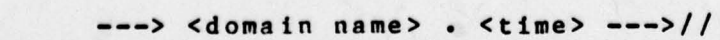
<arith primary>



<number>

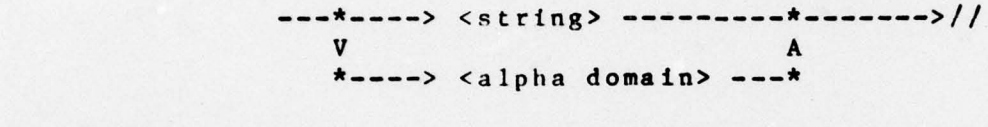


<arith domain>

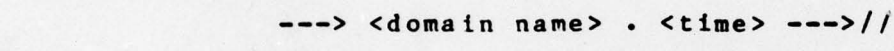


<time>

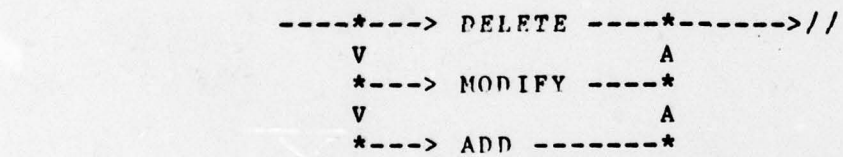
<string exp>



<alpha domain>



<boolean function>



SEMANTICS:

1. ALERT by itself will list the names of all known alerters.
2. ALERT <alert name> will list the text associated with <alert name> and the current state of the alerter.
3. ALERT <alert name> = CANCEL will cancel <alert name>.
4. ALERT <alert name> (<evaluate condition>) = <condition> creates an alerter. <evaluate condition> specifies under what circumstances to evaluate <condition>. When <condition> evaluates true then the <alert name> and transaction number will be displayed at the terminal.
5. When an alerter is being evaluated two tuples can be viewed. These are the tuples before and after the modification. If the transaction being evaluated is a deletion or addition, the two tuples are identical.
6. ALERT <alert name> = ENABLE enables an alerter. An alerter being enabled means it may be triggered. After an alerter is created it is enabled.
7. ALERT <alert name> = DISABLE disables an alerter. An alerter being disabled means it may not be triggered.

<bye command>

SYNTAX:

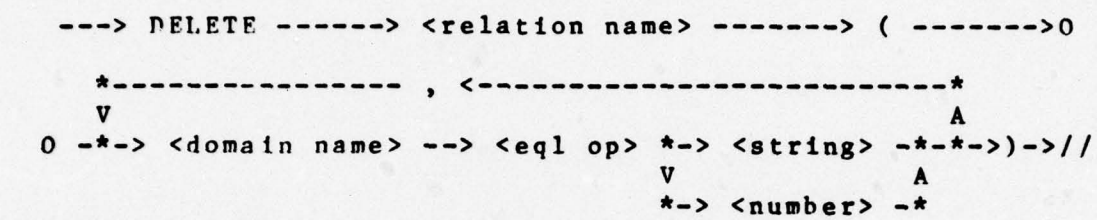
---> BYE --->//

SEMANTICS:

1. The <bye command> causes the end of the session. All changes to the database are saved between sessions. Alerter definitions and states are saved between sessions.

<delete command>

SYNTAX:



SEMANTICS:

1. The <delete command> causes a tuple of <relation name> to be located and deleted from the database. An exception is returned if a tuple does not exist with a key value identical with the key value specified.
2. The default value for any alpha keys not mentioned is blank. The default value for any numeric key not mentioned is zero. Only keys can be used for <domain name> in the <delete command>.
3. Keys of type ALPHA must be compared against <string>s and keys of type NUMBER must be compared against <number>s.
4. There is no current tuple of <relation name> after this command is complete.

<display command>

SYNTAX:

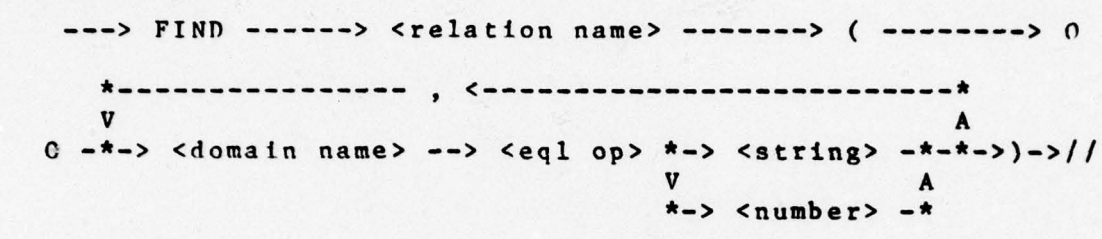
---> DISPLAY ---> <relation name> ----->//

SEMANTICS:

1. The <display command> causes the current tuple of <relation name> to be output. Each line output contains <domain name> and the value of <domain name> in the current tuple.
2. An exception is returned if there is no current tuple of <relation name>.

<find command>

SYNTAX:

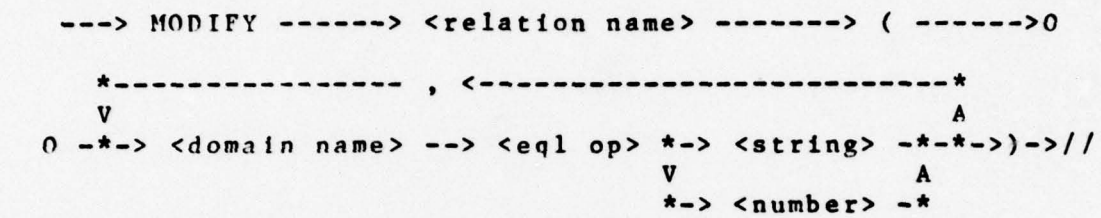


SEMANTICS:

1. The <find command> causes a tuple of <relation name> to be located. An exception is returned if no tuple exists with a key value identical to the specified values.
2. The default value for any alpha keys not mentioned is blank. The default value for any numeric key not mentioned is zero. Only keys can be used for <domain name>.
3. Keys of type ALPHA must be compared against <string>s and keys of type NUMBER must be compared against <number>s.
4. The tuple located becomes the current tuple of <relation name> after this command is complete.

<modify command>

SYNTAX:

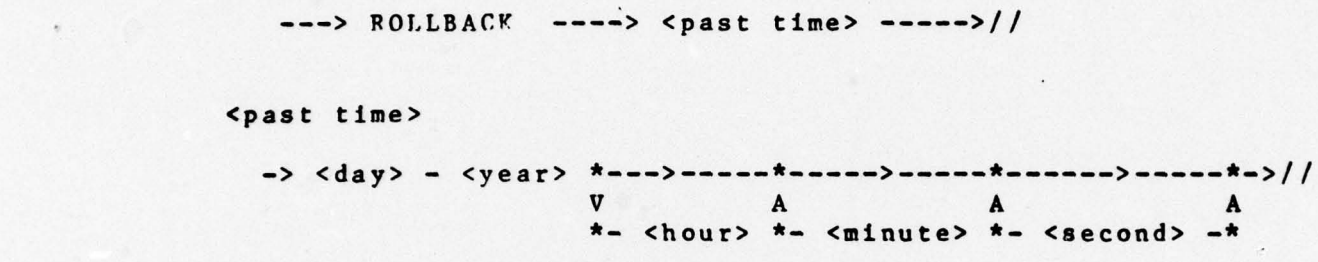


SEMANTICS:

1. The <modify command> causes the current tuple of <relation name> to be modified and to be stored into the database. An exception is returned if there is no current tuple of <relation name>.
2. The keys of a tuple may not be changed. Therefore, only nonkeys may be used in a <modify command>.
3. Domains of type ALPHA must have <string>s assigned to them and domains of type NUMBER must have <number>s assigned to them.
4. The tuple modified remains the current tuple of <relation name>.

<rollback command>

SYNTAX:



SEMANTICS:

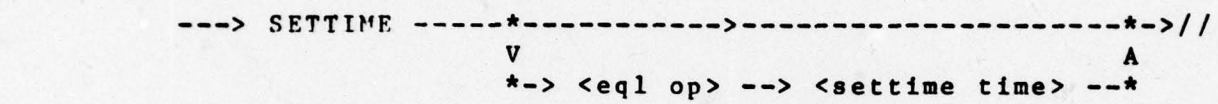
1. ROLLBACK causes the database to be backed up to the point in time specified.

2. This feature is used to back out incorrect changes to the database.

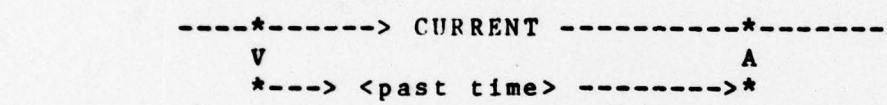
<settime command>

SYNTAX:

<settime command>



<settime time>

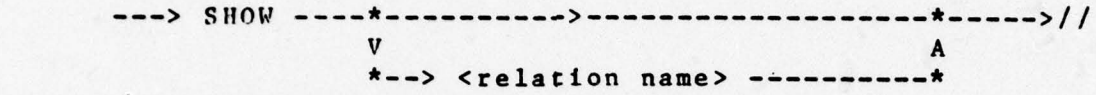


SEMANTICS:

1. SETTIME by itself causes the time from which the database is being viewed to be displayed.
2. SETTIME = CURRENT states the database is being viewed from the current time. This is the normal mode.
3. SETTIME = <day> <year> <hour> <minute> <second> causes the database to be viewed as it existed at that point in time. The add, delete, and modify commands may not be executed until a SETTIME = CURRENT is executed.

<show command>

SYNTAX:



SEMANTICS:

1. SHOW by itself causes the list of <relation name>s to be output.
2. SHOW <relation name> causes the list of <domain name>s within <relation name> to be output. Along with the <domain name> its type, key status, and size are output.

REFERENCES.

[1] Ricardo Cortes. "An Alerting System for a Dynamic Database Management System". University of Pennsylvania Master Thesis. December 1976.

[2] O. Peter Buneman and Howard Lee Morgan. "Alerting in Database Systems:Concepts and Techniques". Department of Decision Sciences, working paper 75-12-02, University of Pennsylvania.

[3] Dennis G. Severance and Guy M. Lohman. "Differential Files:Their Application to the Maintenance of Large Databases". ACM Transactions on Database Systems. September 1976.

[4] James Martin, "Computer Data-base Organization". Prentice-Hall,Inc. 1975.

[5] O. Peter Buneman, Howard Lee Morgan, and Stanley F. Cohen. "Network Alerter Service: Preliminary Design". Department of Decision Sciences, working paper 77-07-03, University of Pennsylvania.

[6] O. Peter Buneman and Howard Lee Morgan, "Implementing

Alerting Techniques in Database Systems". Proceeding of the IEEE Computer Society's First International Computer & Applications Conference. November 1977.

[7] O. J. Dahl, E. W. Dijkstra and C. A. R. Hoare, "Structured Programming", Academic Press, 1972.

[8] B7000/B6000 Series DMSII Inquiry Reference Manual. Burroughs Corporation. 1977.

[9] M. M. Astraham, K. P. Blasgen, D. D. Chamberlin, K. P. Eswaram, J. N. Gray, P. P. Griffiths, W. F. King, R. A. Loire, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson. "System R: Relational Approach to Database Management". ACM Transactions on Database Systems. June,1976.